AMENDMENTS TO THE CLAIMS:

This listing of claims will replace all prior versions and listings of claims in the above-referenced application.

Listing of Claims:

Claims 1 – 34 (Cancelled).

35. (Currently amended) A method of instrumenting an initial byte code computer program, comprising:

(a) examining [[the]] initial byte code to determine an identifier associated with at least one block of code included in the initial byte code;

(b) selecting portions of the initial byte code for instrumentation; and

(c) instrumenting, after a runtime system for executing byte code receives said initial byte code, the portions by supplementing the initial byte code with additional byte code for instrumentation that facilitates runtime data gathering to monitor execution of the computer program, wherein a portion of the additional byte code uses the identifier to instrument portions of the initial byte code associated with said at least one block of code.

36. (Currently amended) The method of claim 35, wherein selecting the portions includes choosing portions of the initial byte code corresponding to at least one of: method entry, method exit, a throw, a method call, and a new line number.

37. (Currently amended) The method of claim 36, wherein instrumenting a portion of the initial byte code corresponding to a method call includes instrumenting a local line number of source code corresponding to the initial byte code being instrumented.

38. (Previously Presented) The method of claim 35, wherein instrumenting the portions includes adding calls to instrumentation runtime functions that pass parameters indicative of the portions being instrumented.

39. (Currently amended) A method of claim 38, wherein at least one of the parameters that is passed includes a line number of [[the]] source code corresponding to the portion being instrumented.

40. (Currently amended) The method of claim 38, wherein at least one of the parameters that is passed includes a thispointer for ~~the~~ a method corresponding to the portion being instrumented.

41. (Previously Presented) The method of claim 38, wherein at least one of the parameters that is passed corresponds to at least one method parameter provided to a method containing byte code that is being instrumented.

42. (Previously Presented) The method of claim 41, wherein data indicative of the at least one method parameter is passed in a message buffer from an instrumentation runtime function to at least one viewer routine that displays the data to a user.

43. (Previously Presented) The method of claim 42, wherein the message buffer includes scalar data, array data, and object data.

44. (Previously Presented) The method of claim 43, further comprising:

(d) placing an object header in the message buffer; and

(e) placing an array header in the message buffer.

45. (Previously Presented) The method of claim 44, further comprising:

(f) limiting the message buffer to a predetermined size.

46. (Previously Presented) The method of claim 38, wherein data indicative of the parameters are stored in a message buffer.

47. (Previously Presented) The method of claim 46, wherein data from the message buffer is passed to at least one viewer routine that displays the data to a user.

48. (Previously Presented) The method of claim 35, further comprising:

(d) instrumenting an end of a method to provide instrumentation for handling an abort.

49. (Previously Presented) The method of claim 35, further comprising:

(d) instrumenting a call to a native function by adding a byte code wrapper to the native function and then instrumenting the wrapper.

50. (Previously Presented) The method of claim 49, wherein the wrapper includes byte code corresponding to method entry and exit portions.

51. (Previously Presented) The method of claim 35, further comprising:

(d) instrumenting a call to a native function by providing an native assembly language thunk that captures data passed to and from the native function.

52. (Previously Presented) The method of claim 51, further comprising:

(e) hooking the assembly language thunk between said runtime system for executing byte code and the call to the native function.

53. (Previously Presented) The method of claim 52, wherein hooking the assembly language thunk includes intercepting a call that provides an address for a procedure.

54. (Previously Presented) The method of claim 35, further comprising:

(d) providing a routine to pass data via a message stream.

55. (Previously Presented) The method of claim 54, further comprising:

(e) providing a data storage to store data provided via the message stream.

56. (Previously Presented) The method of claim 54, further comprising:

(e) providing a viewer to allow viewing at least a subset of data from the message stream as the data is being generated.

57. (Currently amended) A method of instrumenting a computer program, comprising:

(a) examining an initial byte code representation of the program to determine an identifier associated with at least one block of code included in the initial byte code representation;

(b) creating a program counter mapping table corresponding to the initial byte code representation;

(c) selecting portions of the initial byte code representation for instrumenting using the program counter mapping table;

(d) instrumenting, after a runtime system for executing byte code receives said initial byte code representation, the portions by supplementing the initial byte code representation with additional byte code including calls to instrumentation runtime functions at at least some of the portions to facilitate runtime data gathering for monitoring execution of the computer program, wherein a portion of the additional byte code uses the identifier to instrument portions of the initial byte code representation associated with said at least one block of code; and

(e) modifying the program counter mapping table according to original byte code and supplemented byte code.

58. (Previously Presented) The method of claim 57, wherein selecting the portions includes choosing portions of the initial byte code representation corresponding to at least one of: method entry, method exit, a throw, a method call, and a new line number.

59. (Previously Presented) The method of claim 57, wherein instrumenting the portions includes adding calls to instrumentation runtime functions that pass parameters indicative of the portions being instrumented.

60. (Previously Presented) The method of claim 57, further comprising:

(f) instrumenting a call to a native function by adding a byte code wrapper to the native function and then instrumenting the wrapper.

61. (Previously Presented) The method of claim 60, wherein the wrapper includes byte code instrumentation corresponding to method entry and exit portions.

62. (Previously Presented) The method of claim 57, further comprising:

(f) instrumenting a call to a native function by providing a native assembly language thunk that captures data passed to and from the native function.

63. (Previously Presented) The method of claim 62, further comprising:

(g) hooking the assembly language thunk between said runtime system for executing byte code and the call to the native function.

64. (Previously Presented) The method of claim 63, wherein hooking the assembly language thunk includes intercepting a call that provides an address for a procedure.

65. (Previously Presented) The method of claim 57, further comprising:

(f) following examining an initial byte code representation of the program, registering each of the methods and corresponding line numbers thereof with runtime instrumentation code.

66. (Currently amended) The method of claim 65, wherein registering each of the methods and corresponding line numbers thereof facilitates determining [[the]] source code being executed during run time debugging.

67. (Currently amended) A computer program product, stored in a computer-readable medium, that instruments an initial byte code computer program, comprising code that:

(a) examines [[the]] initial byte code to determine an identifier associated with at least one block of code included in the initial byte code;

(b) selects portions of the initial byte code for instrumentation; and

(c) instruments, after a runtime system for executing byte code receives said initial byte code, the portions by supplementing the initial byte code with additional byte code for instrumentation that facilitates runtime data gathering to monitor execution of the computer program, wherein a portion of the additional byte code uses the identifier to instrument portions of the initial byte code associated with said at least one block of code.

68. (Currently amended) The computer program product of claim 67, wherein code that selects the portions includes code that chooses portions of the initial byte code corresponding to at least one of: method entry, method exit, a throw, a method call, and a new line number.

69. (Currently amended) The computer program product of claim 68, wherein code that instruments a portion of the initial byte code corresponding to a method call includes code that instruments a local line number of source code corresponding to the initial byte code being instrumented.

70. (Previously Presented) The computer program product of claim 67, wherein code that instruments the portions includes code that adds calls to instrumentation runtime functions that pass parameters indicative of the portions being instrumented.

71. (Currently amended) The computer program product of claim 70, wherein at least one of the parameters that is passed includes a line number of [[the]] source code corresponding to the portion being instrumented.

72. (Currently amended) The computer program product of claim 70, wherein at least one of the parameters that is passed includes a thispointer for ~~the~~ a method corresponding to the portion being instrumented.

73. (Previously Presented) The computer program product of claim 70, wherein at least one of the parameters that is passed corresponds to at least one method parameter provided to a method containing byte code that is being instrumented.

74. (Previously Presented) The computer program product of claim 73, wherein data indicative of the at least one method parameter is passed in a message buffer from an instrumentation runtime function to at least one viewer routine that displays the data to a user.

75. (Previously Presented) The computer program product of claim 74, wherein the message buffer includes scalar data, array data, and object data.

76. (Previously Presented) The computer program product of claim 75, further comprising code that:

(d) places an object header in the message buffer; and

(e) places an array header in the message buffer.

77. (Previously Presented) The computer program product of claim 76, further comprising code that:

(f) limits the message buffer to a predetermined size.

78. (Previously Presented) The computer program product of claim 70, wherein data indicative of the parameters are stored in a message buffer.

79. (Previously Presented) The computer program product of claim 78, wherein data from the message buffer is passed to at least one viewer routine that displays the data to a user.

80. (Previously Presented) The computer program product of claim 67, further comprising code that:

(d) instruments an end of a method to provide instrumentation for handling an abort.

81. (Previously Presented) The computer program product of claim 67, further comprising code that:

(d) instruments a call to a native function by adding a byte code wrapper to the native function and then instrumenting the wrapper.

82. (Previously Presented) The computer program product of claim 81, wherein the wrapper includes byte code corresponding to method entry and exit portions.

83. (Previously Presented) The computer program product of claim 67, further comprising code that:

(d) instruments a call to a native function by providing an native assembly language thunk that captures data passed to and from the native function.

84. (Previously Presented) The computer program product of claim 83, further comprising code that:

(e) hooks the assembly language thunk between said runtime system for executing byte code and the call to the native function.

85. (Previously Presented) The computer program product of claim 84, wherein the code that hooks the assembly language thunk includes code that intercepts a call that provides an address for a procedure.

86. (Previously Presented) The computer program product of claim 67, further comprising code that:

(d) provides a routine to pass data via a message stream.

87. (Previously Presented) The computer program product of claim 86, further comprising code that:

(e) provides a data storage to store data provided via the message stream.

88. (Previously Presented) The computer program product of claim 86, further comprising code that:

(e) provides a viewer to allow viewing at least a subset of data from the message stream as the data is being generated.

89. (Currently amended) A computer program product, stored in a computer-readable medium, that instruments a computer program, comprising code that:

(a) examines an initial byte code representation of the program to determine an identifier associated with at least one block of code included in the initial byte code representation;

(b) creates a program counter mapping table corresponding to the initial byte code representation;

(c) selects portions of the initial byte code representation for instrumenting using the program counter mapping table;

(d) instruments, after a runtime system for executing byte code receives said initial byte code representation, the portions by supplementing the initial byte code representation with additional byte code including calls to instrumentation runtime functions at at least some of the portions to facilitate runtime data gathering for monitoring execution of the computer program, wherein a portion of the additional byte code uses the identifier to instrument portions of the initial byte code representation associated with said at least one block of code; and

(e) modifies the program counter mapping table according to original byte code and supplemented byte code.

90. (Previously Presented) The computer program product of claim 89, wherein the code that selects the portions includes code that chooses portions of the initial byte code representation corresponding to at least one of: method entry, method exit, a throw, a method call, and a new line number.

91. (Previously Presented) The computer program product of claim 89, wherein the code that instruments the portions includes code that adds calls to instrumentation runtime functions that pass parameters indicative of the portions being instrumented.

92. (Previously Presented) The computer program product of claim 89, further comprising code that:

(f) instruments a call to a native function by adding a byte code wrapper to the native function and then instruments the wrapper.

93. (Previously Presented) The computer program product of claim 92, wherein the wrapper includes byte code instrumentation corresponding to method entry and exit portions.

94. (Previously Presented) The computer program product of claim 89, further comprising code that:

(f) instruments a call to a native function by providing a native assembly language thunk that captures data passed to and from the native function.

95. (Previously Presented) The computer program product of claim 94, further comprising code that:

(g) hooks the assembly language thunk between said runtime system for executing byte code and the call to the native function.

96. (Previously Presented) The computer program product of claim 95, wherein the code that hooks the assembly language thunk includes code that intercepts a call that provides an address for a procedure.

97. (Previously Presented) The computer program product of claim 89, further comprising code that:

(f) following examining an initial byte code representation of the program, registers each of the methods and corresponding line numbers thereof with runtime instrumentation code.

98. (Previously Presented) The computer program product of claim 97, wherein the code that registers each of the methods and corresponding line numbers thereof facilitates determining the source code being executed during run time debugging.